第６回　アセンブラプログラミング　予習資料

第6回授業の前にあらかじめ目を通し、問題を解いておくこと。
今回と次回は、教科書に書かれている命令を組合せて使うことを学ぶ。教科書には個々の命令のあらましが、また配布資料 （http://www.jitec.jp/1_00topic/topic_20081027_hani_yougo.pdf） には個々の命令の動作の具体的な記述があるが、命令を組合せて簡単なプログラムを作ることは触れていないので、資料を中心として勉強する。
（注： 命令は COMET II およびそのアセンブリ言語 CALS II を用いるが、命令セットの異なる他の CPU でも考え方は同じであるし、命令の使い方もそれほど違いが無い。COMET II で慣れておけば、他の CPU への応用も容易なはずである。）

［1］ メモリ上の変数同士を足し合わせて、メモリ上の変数に書き込む

a.　次の CASL プログラム （命令列） は、何をするものか　　（正しいものを選べ）

```
LD    GR3, 100
LD    GR4, 101
ADDA  GR3, GR4
ST    GR3, 102
```

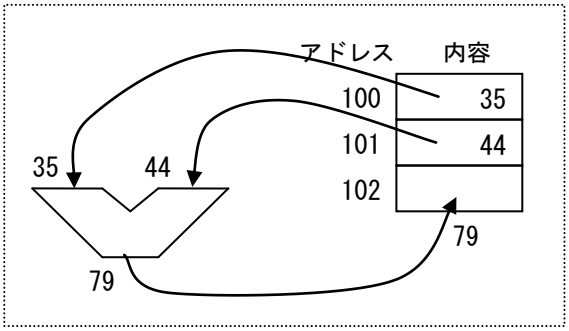
① 汎用レジスタ GR3 の内容をメモリ上 100 番地に、GR4 の内容を 101 番地に退避し、GR3 と GR4 の内容を加算してメモリ上 102 番地に格納する

② メモリ上 100 番地の内容をアドレスと思ってそのアドレスで指されるメモリの内容を汎用レジスタ GR3 に取出し、同様に 101 番地の内容をアドレスと思ってそのアドレスで指されるメモリの内容を GR4 に取出し、GR3 と GR4 を加えて、結果をメモリ上の 102 番地に格納する

③ 数値 100 と 101 を加えて、その結果（201 になる）をメモリ上の 102 番地に格納する

④ メモリ上 100 番地の内容と、101 番地の内容を加算して、メモリ上 102 番地へ格納する

（注意） ここにある LD　GR3, 100 のような命令形式は、「直接アドレッシング」で、≪メモリの 100 番地に入っている中身を LD 命令で汎用レジスタ GR3 に取り出せ≫ という意味である。 気をつけて欲しいのは、GR3 に取り出すのは「値 100」ではなくて、メモリ 100 番地の内容である。 だから、上記のプログラムは、メモリの 100 番地入っている内容と、101 番地に入っている内容を、GR3 と GR4 に取り出して ADDA　GR3, GR4 命令で足し合わせよ、足し合わせた結果が GR3 に入っているので、ST　GR3, 102 の命令で GR3 の内容をメモリの 102 番地に格納せよ、と言っている。
繰返すが、数値 100 と 101 を足して 201 になる、という話ではない。



b.　次の CASL プログラム （命令列） は、何をするものか　　（正しいものを選べ）

```
LD    GR3, 100
ADDA  GR3, 101
ST    GR3, 102
```

① 汎用レジスタ GR3 の内容をメモリ上 100 番地に退避し、GR3 とメモリ上 101 番地の内容を加算してメモリ上 102 番地に格納する

② メモリ上 100 番地の内容をアドレスと思ってそのアドレスで指されるメモリの内容を汎用レジスタ GR3 に取出し、メモリ上 101 番地の内容をアドレスと思ってそのアドレスで指されるメモリの内容を GR3 とを加えて、結果をメモリ上の 102 番地に格納する

③ 数値 100 と 101 を加えて、その結果をメモリ上の 102 番地に格納する

④ メモリ上 100 番地の内容と、101 番地の内容を加算して、メモリ上 102 番地へ格納する

（注意） a)と比較して欲しい。やっていることや結果は、実は同じである。 プログラムというものは、書き方が1つだけというわけではなく、得られる結果が同じになればそれでよいのである。

しかし、両者はまったく同一ではない。 命令の数が異なる。ということは、もしそれぞれの命令を実行するための時間が同じ時間 t であるなら、1－1では実行に 4t かかるのに対し、1－2は 3t で済むので早いことになる。これを1回だけ実行するのだと、差は t だけで、t の実際の値は最近の CPU だと1ナノ秒（＝10億分の1秒）以下だから、体感はできないだろう。もし10億回ぐるぐる回るループの中で1ナノ秒少なくなると、（10億分の1秒）×（10億回）＝1秒の差が出る。

c. メモリ上の 70 番地に変数 X が、80 番地に定数 5 が、75 番地に変数 Y が置かれているとする。 Java/C の文 Y=X+5（変数 X の中身と定数 5 を加えた結果を、変数 Y に書き込む） を実現するアセンブラプログラムはどれか

①
```
ADDA   70  5
ST     75
```
②
```
LD     GR3  5
ADDA   GR3, 70
ST     GR3, 75
```
③
```
LD     GR3, 70
ADDA   GR3, 5
ST     GR3, 75
```
④
```
LD     GR3  70
ADDA   GR3, 80
ST     GR3, 75
```

d. オペランドのアドレスの指定の仕方として、数字 70 （70 番地） を書く代りに、変数の名前 X を書くやり方がある。つまり、 ADDA GR3, X という書き方をする。 X はアセンブラ（コンパイラと同じように、人間が書いたソースコードを機械が実行できる実行可能コード（バイナリコード）に変換するプログラム、但し入力のソースコードが Java や C などの高級言語ではなくて、アセンブリ言語）で、適当な番地（70 番地の代りに、たとえば 1325 番地など）を割当てられて、CPUに対しては ADDA GR3, 1325 というように翻訳される。この書き方を、「変数にラベルをつける（番地を直接書くのでなくラベルで書く）書き方」と呼ぶことがある。

この書き方を使って、

```
LD   GR2, X
ADDA GR2, Y
ST   GR2, Z
```

とすると、このプログラムは何をしているか？

e. 上記の書き方を使って、Java/C の文 Y=X+5 を、自分で書いてみよ。但し、変数 X, Y にはラベル X, Y を、定数 5 にはラベル C5 を付けるものとする。

f. 引き算は、命令 SUBA GR3, X というように書く。 これは汎用レジスタ GR3 の内容から、メモリ上の変数 X の内容を減算し、結果が汎用レジスタ GR3 に入る、という命令である。これを使って、Java/C の文 Y=X-3 の減算の計算をアセンブリ言語で書け。

［2］. LAD 命令の利用

a. 次の命令の意味として正しいものはどれか （正しいものを選べ） （LAD 命令の意味は教科書 3.1.3、配布資料参照）

LAD R3, 1

① メモリ上の 1 番地の内容を、汎用レジスタ R3 に取出す

② 数値 1 を、汎用レジスタ R3 に置く（即値）

③ メモリ上の 1 番地の内容を、汎用レジスタ R3 の値に加えて、R3 に戻す （R3 ← R3 + memory[1]）

④ 数値 1 を、汎用レジスタ R3 の値に加えて、R3 に戻す （R3 ← R3 + 1）（即値）

b. LAD R3, 1 を実行した後の R3 の内容は何か。 LD R3, 1 を実行した後の R3 の内容は何か

c. 汎用レジスタ GR5 に、変数 i の値が入っているものとする。この時、Java/C 言語での i++ （i のインクレメント、i=i+1）に相当する操作は、下記のいずれか

① ADDA GR5 1

② LD GR3 1
ADDA GR5, GR3

③ LDA GR3 1
ADDA GR5, GR3

④ LDA GR3 1
ADDA GR3, GR5

［3］ 比較命令と条件分岐

a. 次の命令の意味として正しいものはどれか （正しいものを選べ）（各命令の意味は教科書 3.1.3、配布資料参照）

CPA R3, R4

① 汎用レジスタ R3 の内容と R4 の内容を比較し、R3>R4 ならば R3 に 1 を、R3=R4 なら R3 に 0 を、R3<R4 なら R3 に −1 を書き込む

② 汎用レジスタ R3 の内容と R4 の内容を比較し、フラグレジスタ FR の SF, ZF ビットを書き込む。SF は R3≧R4 なら 0、R3<R4 なら 1 となり、ZF は R3=R4 なら 1、R3≠R4 なら 0 となる

③ 汎用レジスタ R3 の内容と R4 の内容を比較し、もし R3≧R4 なら次の命令へ進み、R3<R4 なら次の次の命令へ進む

④ 汎用レジスタ R3 の内容と R4 の内容を比較し、もし R3=R4 なら次の命令へ進み、R3≠R4 なら次の次の命令へ進む

b. 次の命令の意味として正しいものはどれか （正しいものを選べ）

JPL L3 （注: L3 はメモリのあるアドレスに付けたラベルとする）

① 無条件に、L3 のアドレスにジャンプする

② L3 で指定されるメモリの内容が正（プラス）であれば次の次の命令へジャンプする。そうでなければ次の命令へ進む

③ フラグレジスタ FR の符号ビット SF が 1 であれば、L3 の番地へジャンプする。そうでなければ次の命令へ進む

④ フラグレジスタ FR の符号ビット SF が 0 かつゼロビット ZF が 0 であれば、L3 の番地へジャンプする、そうでなければ次の命令へ進む

c. 次の命令の意味として正しいものはどれか （正しいものを選べ）

JMI L7 （注: L7 はメモリのあるアドレスに付けたラベルとする）

① 無条件に、L7 のアドレスにジャンプする

② L7 で指定されるメモリの内容が負（マイナス）であれば、次の次の命令へジャンプする。そうでなければ次の命令へ進む

③ フラグレジスタ FR の符号ビット SF が 0 であれば、L7 の番地へジャンプする。そうでなければ次の命令へ進む

④ フラグレジスタ FR の符号ビット SF が 1 であれば、L7 の番地へジャンプする、そうでなければ次の命令へ進む

d. 次のプログラムの意味として正しいものはどれか。但し、変数 C1 は値 1 を保持しているものとする（正しいものを選べ）

CPA GR3, C1
JPL L7 （注: L7 はメモリのあるアドレスに付けたラベルとする）

① 無条件に、L7 のアドレスにジャンプする

② GR3 の内容が 1 より大きければ（GR3>1） L7 の番地へジャンプする。そうでなければ次の命令へ進む

③ GR3 の内容が 1 より大きいか等しければ(GR3≧1) L7 の番地へジャンプする。そうでなければ次の命令へ進む

④ GR3 の内容が 1 より小さければ(GR3<1) L7 の番地へジャンプする。そうでなければ次の命令へ進む

e. 次のプログラムの意味として正しいものはどれか。但し、変数 C1 は値 1 を保持しているものとする(正しいものを選べ)

```
CPA  GR3, C1
JMI  L7                    (注: L7 はメモリのあるアドレスに付けたラベルとする)
```

① 無条件に、L7 のアドレスにジャンプする

② GR3 の内容が 1 より大きければ(GR3>1) L7 の番地へジャンプする。そうでなければ次の命令へ進む

③ GR3 の内容が 1 より小さいか等しければ(GR3≦1) L7 の番地へジャンプする。そうでなければ次の命令へ進む

④ GR3 の内容が 1 より小さければ(GR3<1) L7 の番地へジャンプする。そうでなければ次の命令へ進む

f. 次のプログラムの意味として正しいものはどれか。但し、変数 C1 は値 1 を保持しているものとする (正しいものを選べ)

```
CPA  GR3, C1
JUMP L7                    (注: L7 はメモリのあるアドレスに付けたラベルとする)
```

① 無条件に、L7 のアドレスにジャンプする

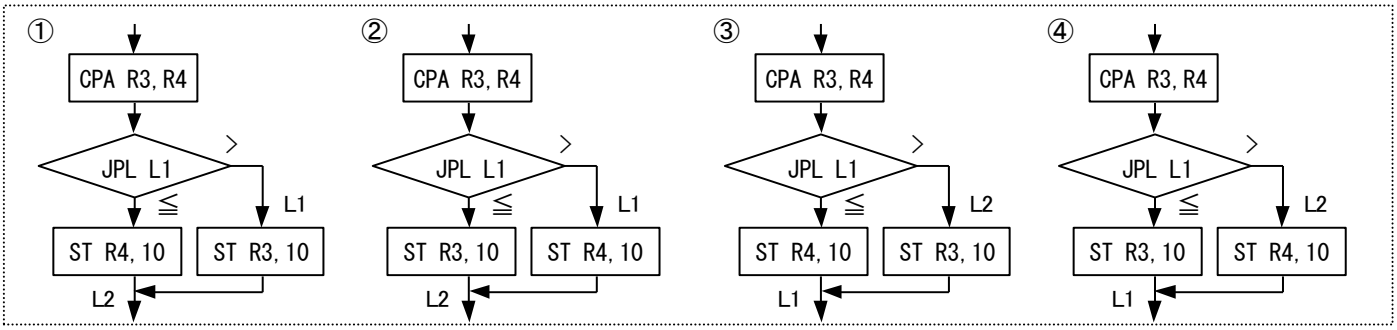
② GR3 の内容が 1 より大きければ(GR3>1) L7 の番地へジャンプする。そうでなければ次の命令へ進む

③ GR3 の内容が 1 より大きいか等しければ(GR3≧1) L7 の番地へジャンプする。そうでなければ次の命令へ進む

④ GR3 の内容が 1 より小さければ(GR3<1) L7 の番地へジャンプする。そうでなければ次の命令へ進む

g. 下記のアセンブリ言語プログラムを正しく表している流れ図を選べ

```
      CPA  R3, R4
      JPL  L1
      ST   R4, 10
      JUMP L2
L1    ST   R3, 10
L2    次の命令
```



h. 上記のアセンブリ言語プログラムを、Java/C 言語に置き換えるとしたら、どれが最も近いか。但し Java/C 言語では汎用レジスタを扱わないので、その代わりに変数 GR3, GR4 に値があるものとし、またメモリ 10 番地の代わりに変数 Z と呼ぶものとする。

① `if (GR3>GR4) { Z = GR4; } else { Z = GR3; }`

② `if (GR3>GR4) { Z = GR3; } else { Z = GR4; }`

③ `if (GR4>GR3) { Z = GR4; } else { Z = GR3; }`

④ `if (GR4>GR3) { Z = GR3; } else { Z = GR4; }`

(注意: たとえば①と③は一見すると同じ意味に見えるが異なる。 GR3=GR4 の場合についてよく考えてみよ)

i. 上記のアセンブリ言語プログラムの(数学的な)意味は何か (正しいものを選べ)

① Z = min(GR3, GR4) 但し等しいときは GR3　　② Z = min(GR3, GR4) 但し等しいときは GR4

③ Z = max(GR3, GR4) 但し等しいときは GR3　　④ Z = max(GR3, GR4) 但し等しいときは GR4

j. 右の図にあるアセンブリ言語プログラムを、Java/C 言語の形に書け。但し、x, p, q は変数とし、C0, C1 は定数(プログラム中で確保)とする。

```
    LD    GR2,x    ; x を GR2 へ
    CPA   GR2,C0   ; 比較(x-0)を計算
    JPL   L1       ; 正なら L1 へ jump
    LD    GR3,q    ; q を GR3 へ
    ADDA  GR3,C1   ; GR3 に定数 1 を足す
    ST    GR3,q    ; GR3 を q へ
    JUMP  L2       ; L2 へ jump
L1  LD    GR3,p    ; p を GR3 へ
    ADDA  GR3,C1   ; GR3 に定数 1 を足す
    ST    GR3,p    ; GR3 を p へ
L2  次の命令
C0  DC    0        ; 定数 0
C1  DC    1        ; 定数 1
```

k. メモリ上の変数 x の「絶対値」を計算するアセンブリ言語プログラムを書け。

(注) 絶対値を計算するには、もし x が負ならば x の符号を反転(-x)し、正か0ならそのまま何もしなくてよい。(0の場合は、符号を反転してしまっても、結果は0になるので、それでも正しい) Java/C 言語で書くと

```
if (x < 0) {
   x = -x;
}
```

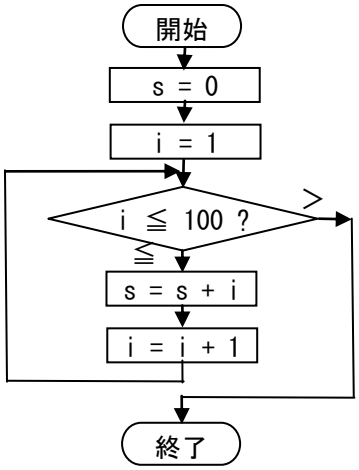
x が負でないときには何もしない(else 節が無い)。

[4] 繰り返し(ループ)へのイントロ　[ループについて追加説明]

繰り返し(ループ)プログラムを考える。Java/C 言語で for 文を使った繰り返しを書いたことがあるだろう。それをアセンブリ言語プログラムに直してみよう。 まずは1から 100 まで加えるプログラムを Java/C で書くと次のようになる。

```
int s = 0;
for (int i=1; i<=100; i++) {
  s = s + i;
}
```

これを、機械命令レベルの流れ図(フローチャート)に書きなおしてみると、右図のようになる。



(ア) s の初期化 s=0 は変数 s に値 0 を代入しなければならない

(イ) for 文の中で、i は 1 に初期化(1 を代入)しなければならない

(ウ) for 文のループと脱出は、右図にあるように、ループの先頭で i<=100 をチェックし、もし>ならば脱出する

(エ) ループの本体は、s = s + i である。これは更に LD、ADDA、ST 命令に分解されるだろうが、ここではそのままにしておく

(オ) ループの最後で i をカウントアップ(i++、つまり i=i+1)しなければならない

(カ) ループの最後から、条件分岐 i<=100 の上へジャンプする。これによってループになる

これを機械命令に書き直せばよいことになる。

a.　右図のプログラムについて、1行ごとに、何をしているかを説明せよ。右側のコメントの部分に、行ごとの番号①、②…を付したので、それごとに書いてみよ。

（注:　変数 S を GR3 に、また変数 I を GR4 に置くものとする。また定数1を GR5 に置いて後で使うものとする。変数 MAX は⑩で1ワード分のメモリを確保し、実行開始前に初期値を 100 にセットされる。）

```
      LAD   GR3,0     ; ① S=0
      LAD   GR4,1     ; ② I=1
      LAD   GR5,1     ; ③ 定数 1 を GR5 に確保
L1    CPA   GR4,MAX   ; ④ I<=MAX?
      JPL   L2        ; ⑤ jump to exit
      ADDA  GR3,GR4   ; ⑥ S=S+I
      ADDA  GR4,GR5   ; ⑦ I=I+1
      JUMP  L1        ; ⑧ jump to top
L2    ST    GR3,S     ; ⑨ S の値をメモリへ
MAX   DC    100       ; ⑩
```

b.　流れ図（フローチャート）を描け

≪解答≫

［1］ a.　④　　メモリの 100 番地の内容を GR3 にコピーし、次にメモリの 101 番地の内容を GR4 にコピーし、ADDA 命令で GR3＋GR4 を計算してその結果を GR3 に格納し、最後に ST 命令で GR3 の内容をメモリの 102 番地にコピーする。

b.　④　　メモリの 100 番地の内容を GR3 にコピーし、次に ADDA 命令で、GR3 の内容とメモリの 101 番地の内容を足してその結果を GR3 に格納し、最後に ST 命令で GR3 の内容をメモリの 102 番地にコピーする。

c.　④

①は、1・2行目とも書き方がそもそもおかしい。COMET Ⅱ では、ADDA 命令も ST 命令も、1つ目のオペランドは汎用レジスタ、2つ目のオペランドは汎用レジスタか又はメモリの実効アドレスである。

②は、メモリ上の5番地の内容を GR3 に LD し、その GR3 とメモリ上の 70 番地の内容を ADDA して結果を GR3 に入れ、最後にその GR3 の内容をメモリ上の 75 番地に書き込んでいる。 1行目の「5番地から持ってくる」がおかしいことになる。

③は、②と同様に、2行目の ADDA の2番目のオペランドが「5番地の内容と加算する」になっていて、おかしい。

（注意:LD, ST, ADDA 命令のいずれも、オペランドに書かれた値はメモリ（か汎用レジスタ）のアドレスを示す。だから定数5をオペランドに書くことはできない。）

d.

LD 命令でメモリ上のラベル X を付けた場所 （＝変数 X に当たる） の内容を GR2 へロードし、
ADDA 命令でその GR2 とメモリ上のラベル Y を付けた場所(変数 Y)の内容とを足し合わせて結果を GR2 へ置き、
ST 命令でその結果の GR2 の内容をメモリ上のラベル Z を付けた場所(変数 Z)へ格納する。
つまり、 Z ＝ X ＋ Y が実現されている。

e.
```
LD    GR3, X
ADDA  GR3, C5
ST    GR3, Y
```
（注） 作業場所として使った GR3 は、他の汎用レジスタでもよい。ただ、この3つの命令の処理に使うので内容を書換えてしまうから、以前に使っていた GR3 の内容をこの3つの命令の後で使いたいのであれば、GR3 ではなくて他の「空いている」汎用レジスタを使うべきである。

f.
```
LD    GR3, X
SUBA  GR3, C5
ST    GR3, Y
```
[2]

a. ② （注意） COMET では LAD 命令のみが「即値」(オペランドに書かれた数値そのものを値とする)である。

b. LAD R3, 1 は、R3 に即値で値1をロードする。R3 の内容は値1。
LD R3, 1 は、R3 に直接アドレシングで1番地を読出した値をロードする。R3 の内容はメモリ上の1番地の中身。

c. ③ （注意: ①や②はメモリの1番地の内容を読出して GR5 に加える。 ④は結果が GR5 ではなくて GR3 に残る）

[3]

a. ② （注意） R3 と R4 の順序は大小比較の向きと関わる。 CPA 命令は SUBA と同じような動作をするが、引き算の結果をレジスタに書き込まない。 つまり、CPA R3, R4 は R3－R4 の結果の正負をフラグレジスタに残すと考えてよい。オペランドの順序が逆の CPA R4, R3 だと、R4－R3 の結果の正負を判定することになる。

b. ④ （注意） COMET のフラグレジスタのビットの定義は、（教科書 2.1.4 のフラグレジスタの項を参照）
SF は比較＝引き算＝した結果の符号ビットを保持し
ZF は結果がオール零であるかどうかを保持する。
但し SF 符号ビットは、2の補数の左端ビットで、正か零のときに0、負のときに1である。
つまり、結果を正・零・負に分けると、正の時は SF=0 かつ ZF=0、零の時は ZF=1(かつ SF=0)、負のときは SF=1(かつ ZF=0)である。
JPL は「プラスであればジャンプ」なので、SF=0 かつ ZF=0 でを条件(つまり正でかつゼロでない時)にジャンプする)

c. ④ （上記と同じ説明）

d. ② （CPA 命令は、GR3－C1 を計算し、その結果 SF ビットと ZF ビットをセットする。JPL 命令は GR3－C1 が正であれば(SF=1 かつ ZF=0)ジャンプする。）

e. ④ JMI は、GR3－C1 が「マイナスであればジャンプ」

f. ① JUMP 命令は、無条件ジャンプである。だから、1行目にある CPA R3, R4 の結果に関わらず L7 へジャンプする。

g. ① 条件分岐 JPL L1 の分岐方向とその行先ラベル、更にそのラベルの先にある命令に注意をして選ぶこと

h. ② i. ④ j. if (x>0) p=p+1; else q=q+1;

k.
```
LDA GR4, 0      (GR4 に即値 0 を代入)
CPA GR4, x      (GR4-x の正負判定)
JPL L1          ((GR4-x)が正 （＝x-0 が負か零） なら L1 へジャンプ)
LDA GR4, 0      (やらなくてもいい。前の値 0 が残っている)
```

SUB GR4, x　　(GR4←(GR4-x)を計算、GR4 は-x の値になる)

ST　GR4, x　　(-x の値を変数 x に代入)

L1　次の命令

[4]

a.

① 即値ロード(LAD)命令で、GR3 に値 0 を入れる。　GR3 は変数 S として使う。 つまり S = 0 ;

② 即値ロード(LAD)命令で、GR4 に値 1 を入れる。　GR4 は変数 I として使う。 つまり I = 0 ;

③ 即値ロード(LAD)命令で、GR5 に値 1 を入れる。　GR5 は定数 1 として使う。

④ 比較命令 CPA で、GR4(= I)と MAX を比較。 つまり、次の JPL 命令と組になって if ((I -MAX) > 0) をする。

⑤ ④の結果、I－MAX が正なら、ラベル L2 の行へ飛ぶ。 正でなければ次の行へ移る。

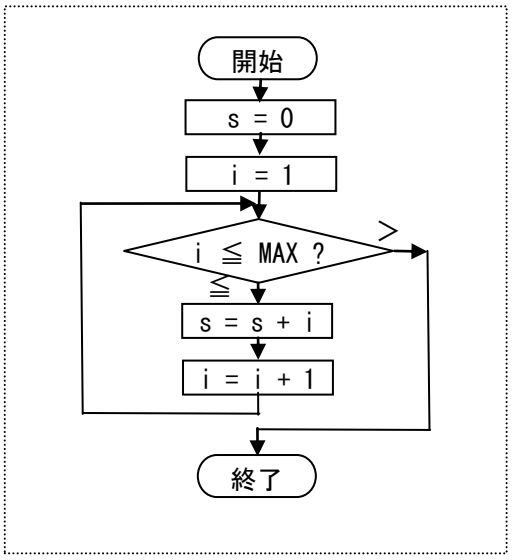
⑥　(正でない時) GR3(=S)＋GR4(=I) を計算して GR3(=S)へ書き戻す。 つまり S = S + I;

⑦　GR4 (=I)＋GR5(=1) を計算して GR4(=I)へ書き戻す。 つまり I = I + 1;

⑧　ループの先頭へ戻る

⑨ (正の時～I>MAX の時) ST 命令で GR3 の内容 (S の加算結果)を、メモリ上の変数領域 S にコピーする

b.　流れ図(フローチャート)に描くと下のようになる。これから、s ＝ Σ(i) を、i＝1からi＝MAXまで足し合わせていることが分かる。(i＝MAXを含む)



［補足］ アセンブリ言語プログラムを実際に動かしてみよう。

この授業で取上げたプロセッサ COMET II は仮想のもので、実在はしない。シミュレータ上で動かすことになる。（シミュレータとは、プロセッサのハードウェアの動きをプログラムとして実現したもの） フリー（無料）の COMET II のシミュレータはいくつか提供されており、ネットからダウンロード・インストールして利用できる。 ここで紹介する（実際に使ってみる）オンライン型のシミュレータ （ http://www.officedaytime.com/dcaslj/index.html ）はインストールの必要が無いので手軽に試せる。使い方は、上記ページをアクセスし、①ソースウィンドウに CASL II で書いたソースプログラムをコピー＆ペーストし、アセンブルボタンを押す、②アセンブルできたら、  ボタンを押して「ステップ実行」（1命令ずつ実行すること、本実行ボタン＝黒三角のボタン＝を押すとプログラムスタートしたように一気に実行するので様子が分からない） する。

アセンブラプログラムの書き方（追加の注意）： 今までのアセンブラプログラムの書き方は便宜的なもので、きちんとアセンブラに処理させるにはいくつか守らなければならないルールがある。

```
TEST START                    ← START アセンブラ命令が必要（ア）＋ラベルが必要（イ）
     CPA   GR3,GR4            ← 命令の位置を縦に揃える（ウ）、汎用レジスタは GRn（オ）、（キ）
     JPL   L1                 ← ラベルは英大文字から始まる英数字で1～8文字（エ）
     ST    GR4,10             ← メモリオペランドに数字を書いたらそのアドレス（カ）、（キ）
     JUMP L2
L1   ST    GR3,10             ← 2つのオペランドをコンマで区切るとき空白を空けない（キ）
L2   NOP                      ← NOP は何もしない命令
     END                      ← 最後に END アセンブラ命令が必要（ア）
```

（ア） プログラムの先頭に START アセンブラ命令（擬似命令）、最後に END アセンブラ命令（擬似命令）を置く

（イ） 先頭の START アセンブラ命令には、プログラムの名前を表すラベルが必要 （上記の例では TEST）

（ウ） 命令部分（アセンブラ命令を含む）の縦位置は揃える （注： これは必須ではないのだが、見やすくするために是非守りたい。本当に必要なのは、ラベルを書かない行で（ラベルがないことを示す）先頭の1つ以上の空白である。

（エ） ラベルは英大文字から始まる英数字で、1～8文字

（オ） 汎用レジスタは GR0～GR8 で表す。これらは予約語であり、ラベルとして使ってはならない

（カ） メモリオペランドに数字を書いたらアドレス、ラベルを書いたらそのラベルの語の置かれるアドレス

（キ） 2つ以上のオペランドをコンマで区切って書くとき、コンマの前後に空白を開けてはいけない

このプログラム例だと、GR3 と GR4 にあらかじめ値が入っていることになっている（プログラム内では値をセットしていない）。そこで実行開始前に、シミュレータ画面上の GR3 と GR4 にそれぞれ 0003, 0005 をセットする。その後、ステップ実行する。なお、ステップ実行する前は、右下に出ているプログラムの黄色い帯が 0000 CPA GR3,GR4 の上にある（今から実行しようとする命令が CPA 命令であることを示す）ものが、ステップ実行すると次の 0001 JPL L1 の上に来るはずである。

1回ステップ実行すると、PR（プログラムカウンタレジスタ）が 0000 から 0001 になり、SF（符号フラグ）が 0 から 1 になる。ZF（ゼロフラグ）は 0 のままである。つまり、GR3 の内容 3 と GR4 の内容 5 を比較した結果、GR3<GR4 であるから CPA 命令の動作記述どおりに、SF=1、ZF=0 にセットされたわけである。

もう1回ステップ実行ボタンを押して、次の命令を実行する。 次の命令は JPL L1 であるから、もし前の CPA の結果が正（SF=0 かつ ZF=0）なら L1 へジャンプするが、そうでなければ次の命令へ進むはずである。 ステップ実行した結果は、すぐ下の 0003 ST GR4, 10 の上に来る。なぜなら、条件分岐命令 JPL の条件を満たさなかった（SF が 1 だった）からである。

もう1回ステップ実行ボタンを押すと、次の 0005 JUMP L2 命令に移り、更にもう1度押すと無条件ジャンプして 0009 L2 NOP へ移る。（0007 L1 ST GR3,10 の行は実行しないことに注意）

ここまで到達すると、GRn の欄より1段下側の欄のメモリ表示（0000, 0004, 0008, 000C と書いてある）の中で、メモリの 10 番地（000A 番地）の内容が 0005 になっているはず（10 番地に max(R3, R4)を代入したはず）なので、確認して欲しい。

以下、足し算をする例をシミュレータ上で実行した例を、画面コピーで示す

```
ADD     START
;
        LD      GR4,CTHREE
        LD      GR3,CFIVE
        ADDA    GR4,GR3
        ST      GR4,RESULT
        RET
;
CTHREE  DC      3
CFIVE   DC      5
RESULT  DS      1
        END
```

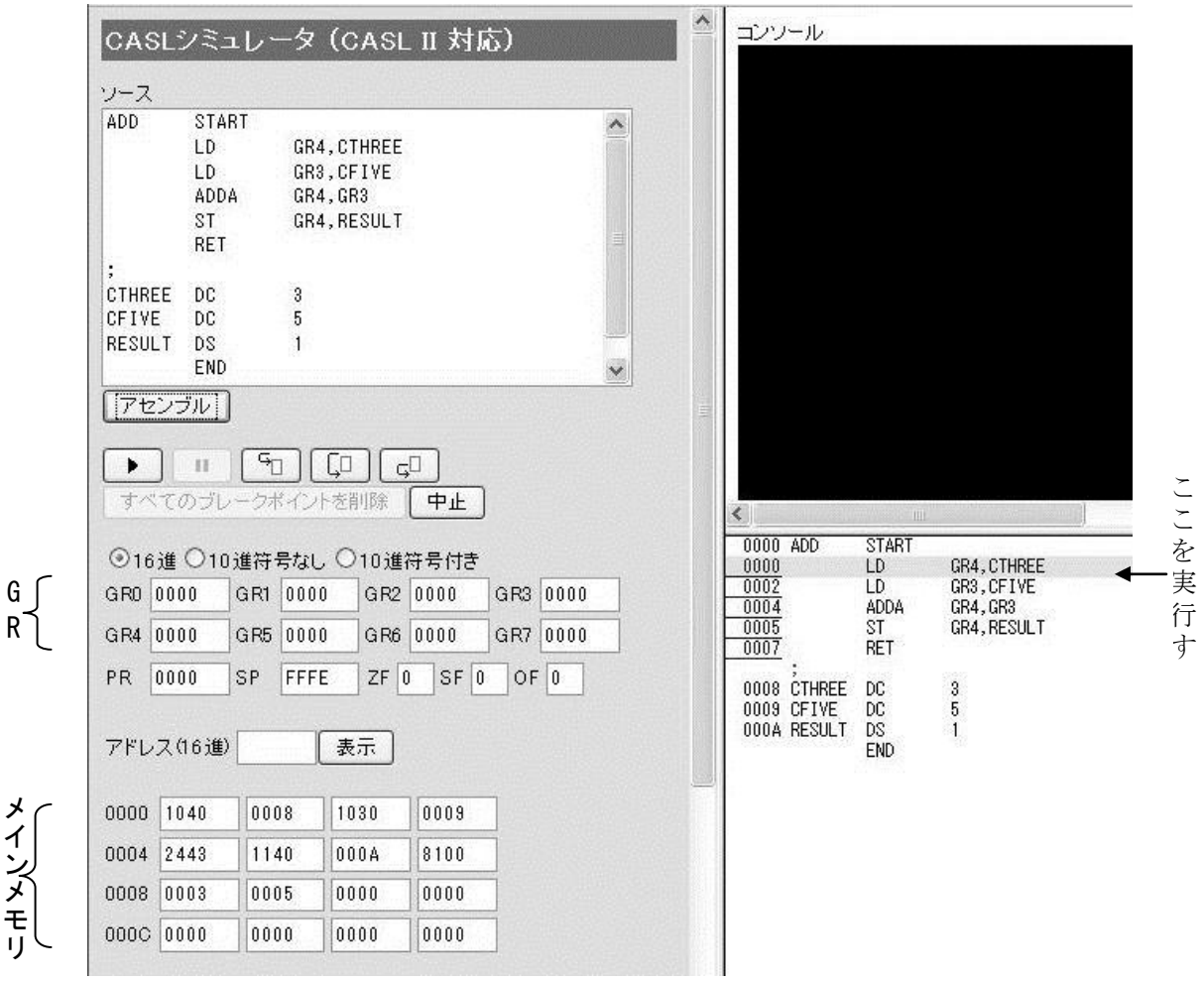
```
        0000    LD    GR4,0008    1040  0008

        0002    LD    GR3,0009    1030  0009

        0004    ADDA  GR4,GR3     2443
        0005    ST    GR4,RESULT  1140  000A

        0007    RET               8100
CTHREE  0008    DC                0003
CFIVE   0009    DC                0005
RESULT  000A    DS                0000
```

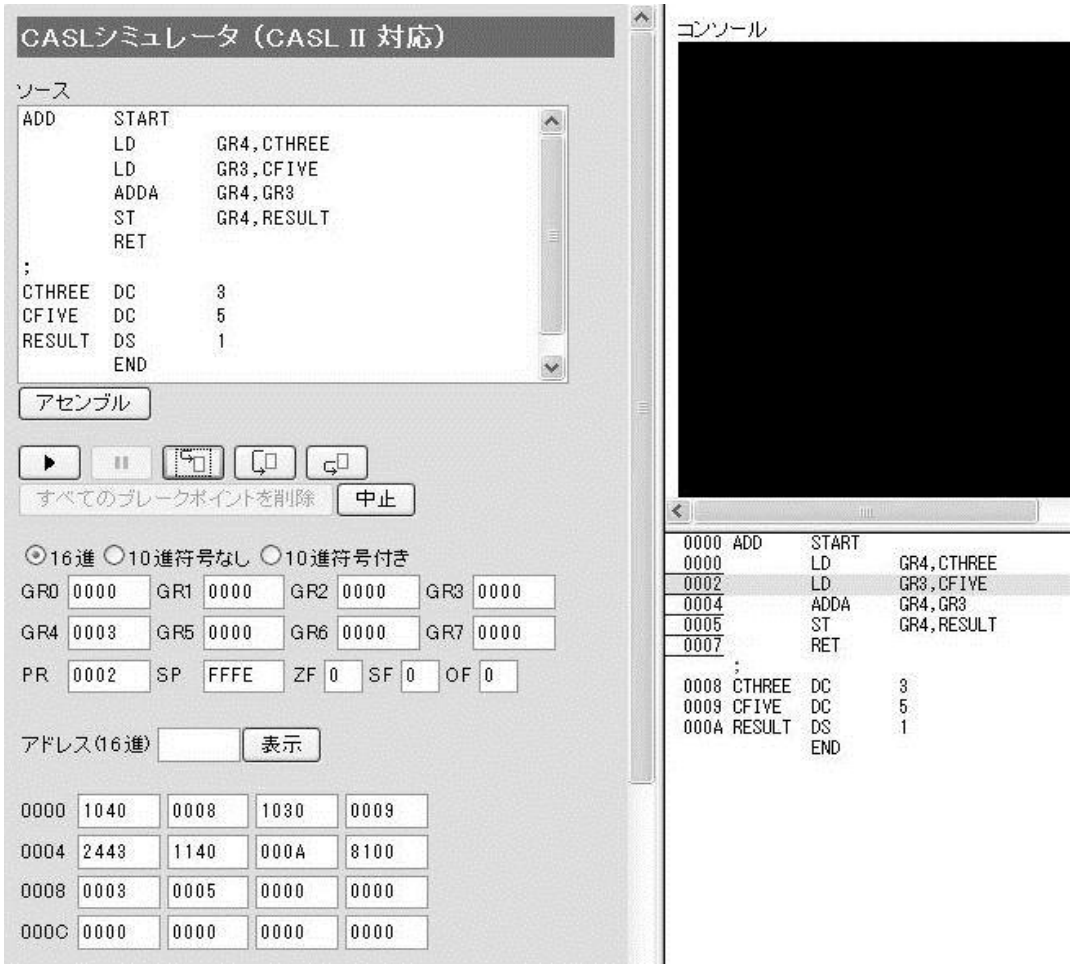
LD 命令 ⇒ 第1語: OP コードが 10(LD 命令、adr あり)、 r/r1 部分が 4(r が GR4)、x/r2 部分は 0(x 指定なし)
第2語: adr が 0008 番地(CTHREE を置いた場所)

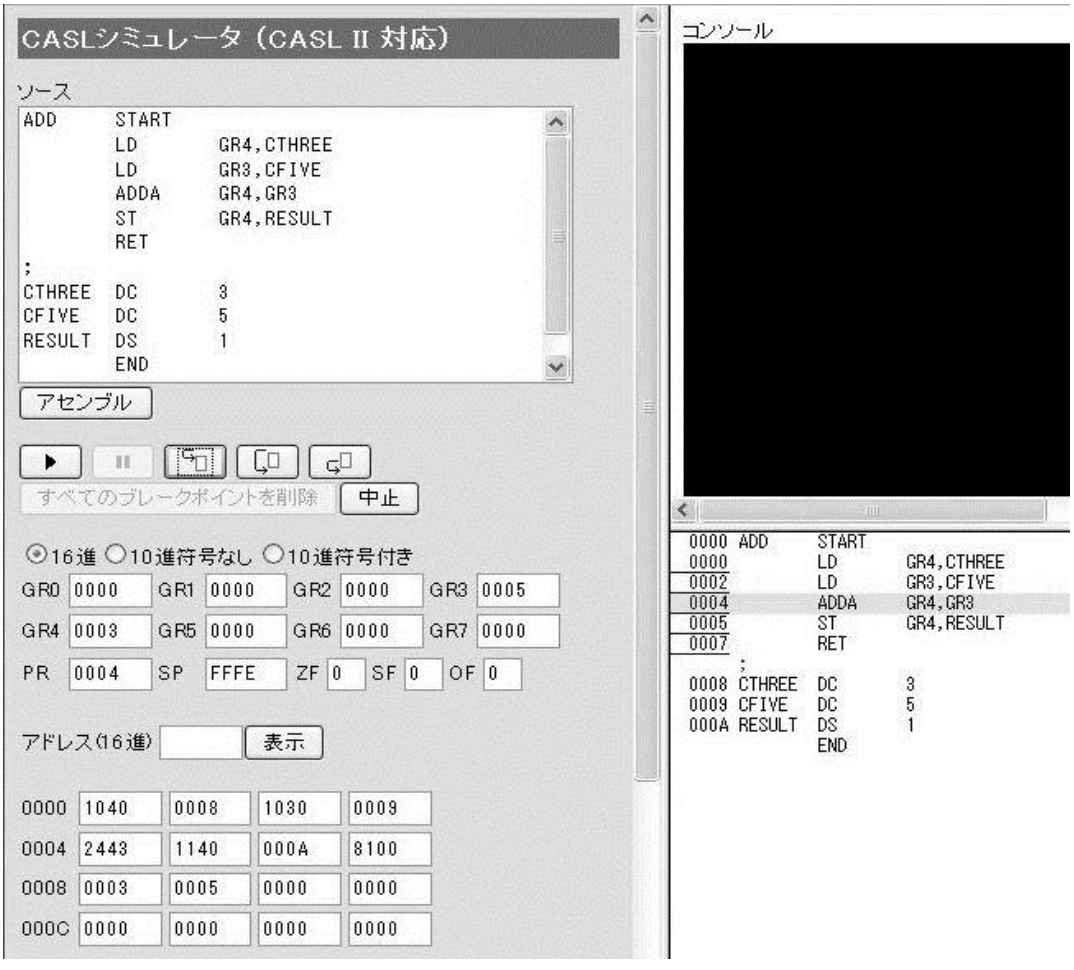
LD 命令 ⇒ 第1語: OP コードが 10(LD 命令、adr あり)、 r/r1 部分が 3(r が GR3)、x/r2 部分は 0(x 指定なし)
第2語: adr が 0009 番地(CFIVE を置いた場所)

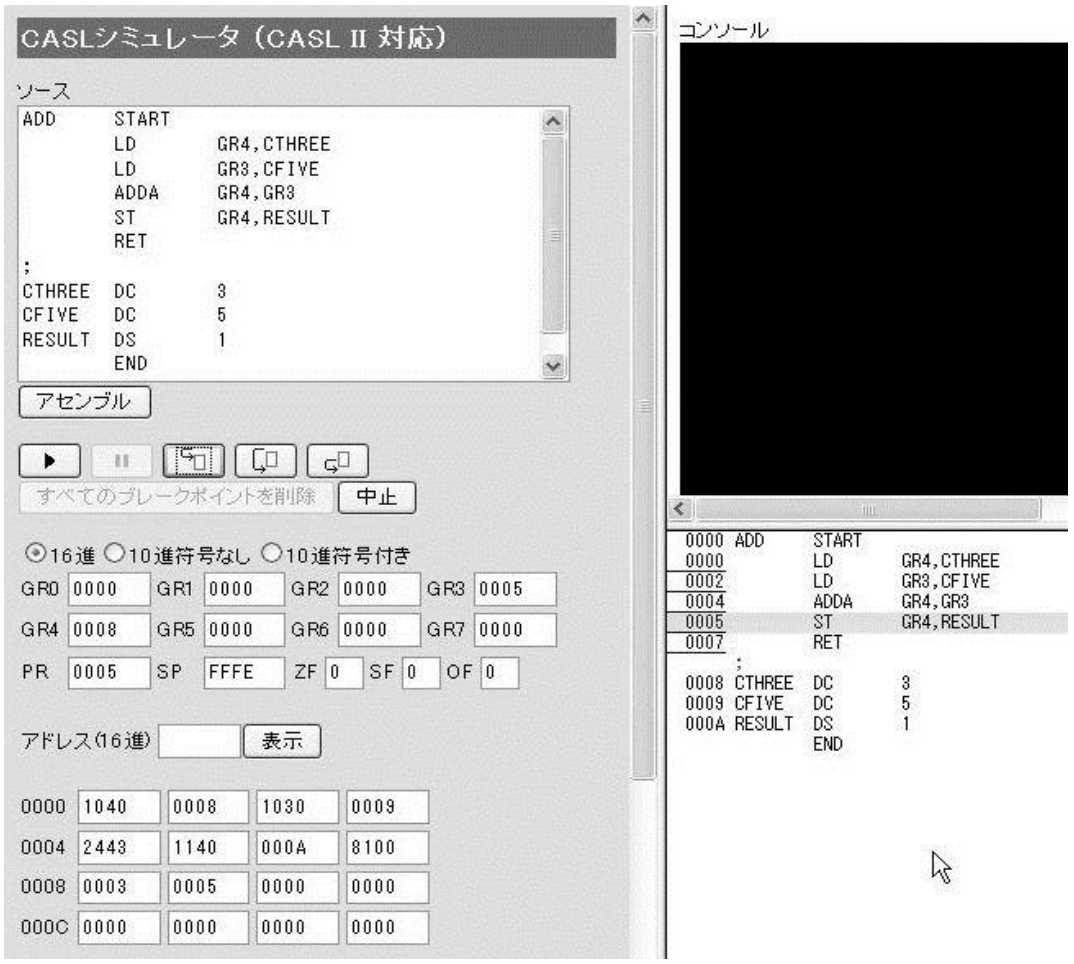
ADDA 命令 ⇒ 第1語: OP コードが 24(ADDA 命令、「r1,r2」形式)、 r/r1 部分が 4(GR4)、x/r2 部分は 3(GR3)

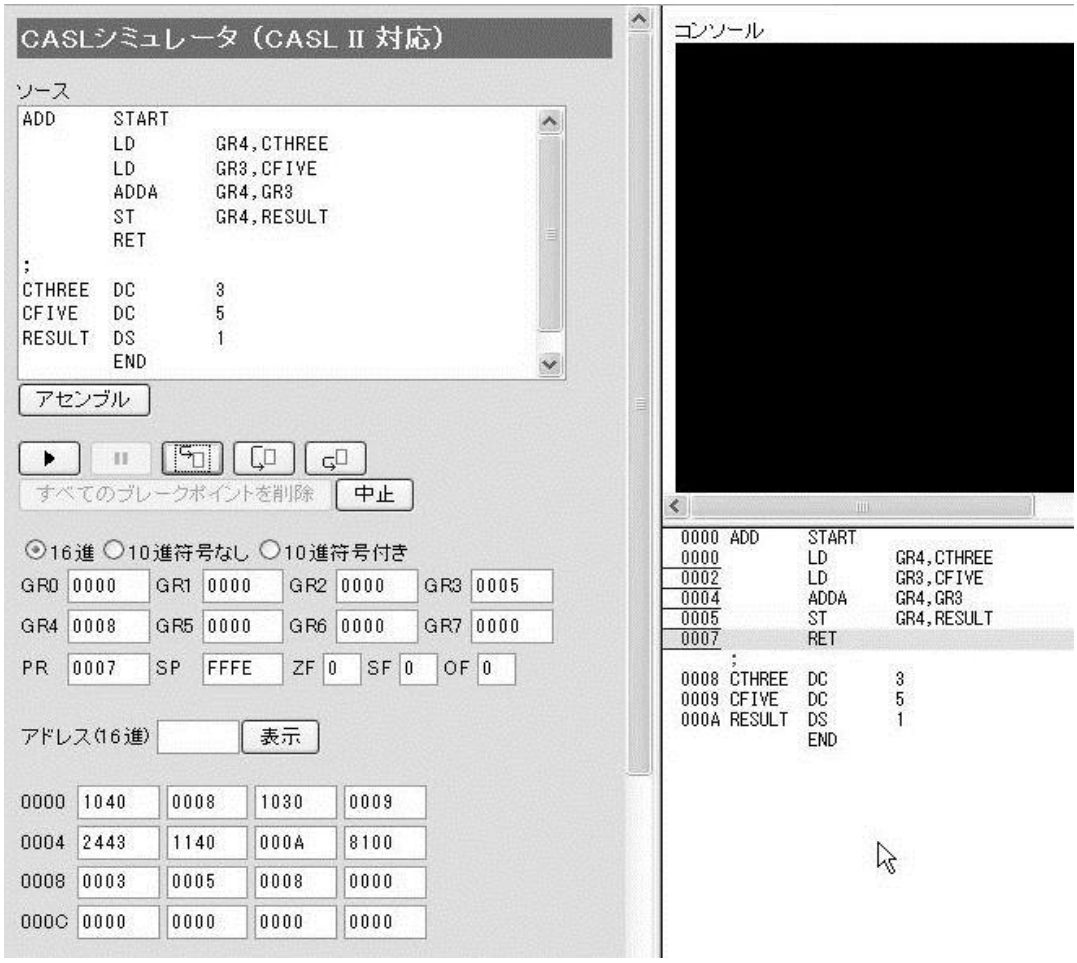
以下略


CASLシミュレータ（CASL II 対応）
ソース
ADD START
LD GR4,CTHREE
LD GR3,CFIVE
ADDA GR4,GR3
ST GR4,RESULT
RET
;
CTHREE DC 3
CFIVE DC 5
RESULT DS 1
END
アセンブル
すべてのブレークポイントを削除
中止
16進 10進符号なし 10進符号付き
GR0 0000 GR1 0000 GR2 0000 GR3 0000
GR4 0000 GR5 0000 GR6 0000 GR7 0000
PR 0000 SP FFFE ZF 0 SF 0 OF 0
アドレス(16進)
表示
0000 1040 0008 1030 0009
0004 2443 1140 000A 8100
0008 0003 0005 0000 0000
000C 0000 0000 0000 0000
コンソール
0000 ADD START
0000 LD GR4,CTHREE
0002 LD GR3,CFIVE
0004 ADDA GR4,GR3
0005 ST GR4,RESULT
0007 RET
;
0008 CTHREE DC 3
0009 CFIVE DC 5
000A RESULT DS 1
END
GR
メインメモリ
ここを実行す


CASLシミュレータ（CASL II 対応）
ソース
ADD START
LD GR4,CTHREE
LD GR3,CFIVE
ADDA GR4,GR3
ST GR4,RESULT
RET
;
CTHREE DC 3
CFIVE DC 5
RESULT DS 1
END
アセンブル
すべてのブレークポイントを削除
中止
16進 10進符号なし 10進符号付き
GR0 0000 GR1 0000 GR2 0000 GR3 0000
GR4 0003 GR5 0000 GR6 0000 GR7 0000
PR 0002 SP FFFE ZF 0 SF 0 OF 0
アドレス(16進)
表示
0000 1040 0008 1030 0009
0004 2443 1140 000A 8100
0008 0003 0005 0000 0000
000C 0000 0000 0000 0000
コンソール
0000 ADD START
0000 LD GR4,CTHREE
0002 LD GR3,CFIVE
0004 ADDA GR4,GR3
0005 ST GR4,RESULT
0007 RET
;
0008 CTHREE DC 3
0009 CFIVE DC 5
000A RESULT DS 1
END


CASLシミュレータ（CASL II 対応）
ソース
ADD START
LD GR4,CTHREE
LD GR3,CFIVE
ADDA GR4,GR3
ST GR4,RESULT
RET
;
CTHREE DC 3
CFIVE DC 5
RESULT DS 1
END
アセンブル
すべてのブレークポイントを削除
中止
16進 10進符号なし 10進符号付き
GR0 0000 GR1 0000 GR2 0000 GR3 0005
GR4 0003 GR5 0000 GR6 0000 GR7 0000
PR 0004 SP FFFE ZF 0 SF 0 OF 0
アドレス(16進)
表示
0000 1040 0008 1030 0009
0004 2443 1140 000A 8100
0008 0003 0005 0000 0000
000C 0000 0000 0000 0000
コンソール
0000 ADD START
0000 LD GR4,CTHREE
0002 LD GR3,CFIVE
0004 ADDA GR4,GR3
0005 ST GR4,RESULT
0007 RET
;
0008 CTHREE DC 3
0009 CFIVE DC 5
000A RESULT DS 1
END


CASLシミュレータ（CASL II 対応）
ソース
ADD START
LD GR4,CTHREE
LD GR3,CFIVE
ADDA GR4,GR3
ST GR4,RESULT
RET
;
CTHREE DC 3
CFIVE DC 5
RESULT DS 1
END
アセンブル
すべてのブレークポイントを削除
中止
16進 10進符号なし 10進符号付き
GR0 0000 GR1 0000 GR2 0000 GR3 0005
GR4 0008 GR5 0000 GR6 0000 GR7 0000
PR 0005 SP FFFE ZF 0 SF 0 OF 0
アドレス(16進)
表示
0000 1040 0008 1030 0009
0004 2443 1140 000A 8100
0008 0003 0005 0000 0000
000C 0000 0000 0000 0000
コンソール
0000 ADD START
0000 LD GR4,CTHREE
0002 LD GR3,CFIVE
0004 ADDA GR4,GR3
0005 ST GR4,RESULT
0007 RET
;
0008 CTHREE DC 3
0009 CFIVE DC 5
000A RESULT DS 1
END


CASLシミュレータ（CASL II 対応）
ソース
ADD START
LD GR4,CTHREE
LD GR3,CFIVE
ADDA GR4,GR3
ST GR4,RESULT
RET
;
CTHREE DC 3
CFIVE DC 5
RESULT DS 1
END
アセンブル
すべてのブレークポイントを削除
中止
16進 10進符号なし 10進符号付き
GR0 0000 GR1 0000 GR2 0000 GR3 0005
GR4 0008 GR5 0000 GR6 0000 GR7 0000
PR 0007 SP FFFE ZF 0 SF 0 OF 0
アドレス(16進)
表示
0000 1040 0008 1030 0009
0004 2443 1140 000A 8100
0008 0003 0005 0008 0000
000C 0000 0000 0000 0000
コンソール
0000 ADD START
0000 LD GR4,CTHREE
0002 LD GR3,CFIVE
0004 ADDA GR4,GR3
0005 ST GR4,RESULT
0007 RET
;
0008 CTHREE DC 3
0009 CFIVE DC 5
000A RESULT DS 1
END

≪基本情報技術者試験　アセンブリプログラム(午後の部)からの例≫

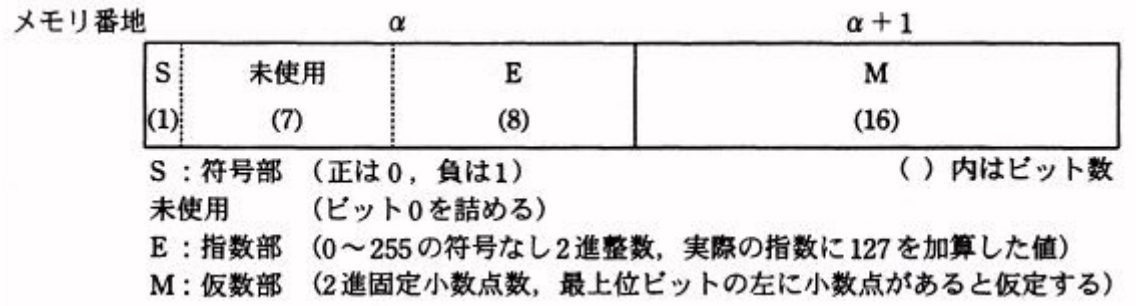
参考までに、基本情報の午後の部の CASL（アセンブラプログラミング） の問題を挙げておく。 かなり面倒だと思うが、知識としては知っていることばかりで解けると思う。 挑戦してみたらどうだろうか。

（基本 22 春午後 12）　次のアセンブラプログラムの説明及びプログラムを読んで，設問1，2に答えよ。

〔プログラムの説明〕

浮動小数点数の加算を行う副プログラム FADD である。

(1) 浮動小数点数は，メモリ中の連続する2語に次の形式で格納する。



この形式で表される浮動小数点数は (－1)S × 0.M × 2E-127 である。 S＝E＝M＝0 でゼロを表現する(M＝0 のときは，必ず S＝E＝0 であること)。

(2) FADD は，被加数と加数について，いずれか指数の大きい方にもう一方の指数をそろえてから仮数を加算し，結果がゼロの場合を除き正規化(仮数部の最上位ビットが1となるように指数部を調整)する。指数部調整の過程で，アンダフロー(指数部が0未満になるので正規化が不可能)，オーバフロー(指数部が 255 を超えるので正規化が不可能)は発生しないものとする。

正規化の例

〔E = 10000100　　M = 0010000000000000〕
↓正規化 (M を 2 ビット左に論理シフトし，E から 2 を減じる)
〔E = 10000010　　M = 1000000000000000〕

(3) FADD は，被加数 X，加数 Y 及び結果 Z の格納領域の先頭番地を，それぞれ GR1，GR2，GR3 に設定して呼び出される。

(4) FADD から戻るとき，汎用レジスタ GR1 ～ GR7 の内容は元に戻す。

〔プログラム〕

(行番号)

```
 1  FADD    START                ; Z ← X ＋ Y
 2          RPUSH
 3          PUSH   0,GR3         ; 結果 Z の格納領域の先頭番地を退避
 4          LD     GR4,0,GR1
 5          AND    GR4,=#00FF    ; Ex: X の指数
 6          LD     GR5,0,GR2
 7          AND    GR5,=#00FF    ; Ey: Y の指数
 8          LD     GR6,1,GR1     ; Mx: X の仮数
 9          LD     GR7,1,GR2     ; My: Y の仮数
10  ; 加算前の準備 (指数をそろえる)
11  ; GR4 ← max(Ex,Ey) , GR6 ← 調整済 Mx , GR7 ← 調整済 My
12          LD     GR3,GR4
13          CPL    GR4,GR5
14          JZE    MADD          ; Ex ＝ Ey の場合
```

```
15            JMI     BIGEY           ; Ex ＜ Ey の場合
16            SUBL    GR3,GR5
17            SRL     GR7,0,GR3       ; My を調整
18            JUMP    MADD
19  BIGEY     [ a ]
20            SUBL    GR5,GR3
21            SRL     GR6,0,GR5       ; Mx を調整
22  ; 符号を考慮した仮数の加算
23  ; Sz: Z の符号 , Ez: Z の指数 , Mz: Z の仮数
24  ; GR4 ← (Sz,Ez) , GR5 ← Mz
25  MADD      LD      GR1,0,GR1       ; X の符号の検査
26            JMI     XMINUS          ;    負の場合
27            LD      GR2,0,GR2       ; Y の符号の検査
28            [ b ]
29            LD      GR5,GR6         ; X ≧ 0 , Y ≦ 0 の場合
30            SUBL    GR5,GR7         ; Mz ← 調整済 Mx ー 調整済 My
31            JUMP    SCHECK
32  XMINUS    LD      GR2,0,GR2
33            JMI     YMINUS
34            LD      GR5,GR7
35            SUBL    GR5,GR6
36            [ c ]
37  YMINUS    OR      GR4,=#8000      ; Z に負符号を設定
38  ADDMXY    LD      GR5,GR6
39            ADDL    GR5,GR7         ; Mz ← 調整済 Mx ＋ 調整済 My
40            JOV     ADJST           ; けた上がりがある場合の正規化
41            JUMP    NORM
42  SCHECK    JOV     NEGMZ           ; Mz の符号を検査
43            JUMP    NORM
44  NEGMZ     OR      GR4,=#8000      ; Sz に負符号を設定
45            XOR     GR5,=#FFFF      ; Mz ← ーMz
46            ADDL    GR5,=1
47  ;加算結果の正規化
48  NORM      LD      GR5,GR5         ; ゼロチェック
49            JNZ     LOOP
50            LD      GR4,=0
51            JUMP    FIN
52  LOOP      LD      GR5,GR5         ; 正規化完了？
53            JMI     FIN
54            [ d ]
55            SUBL    GR4,=1
56            JUMP    LOOP
57  ADJST     SRL     GR5,1
58            OR      GR5,=#8000      ; Mz の最上位ビットを 1 に設定
59            [ e ]
60  FIN       POP     GR3
61            ST      GR4,0,GR3       ; 結果 Z の格納
62            ST      GR5,1,GR3
63            RPOP
64            RET
65            END
```

設問1 プログラム中の［ a ～ e ］に入れる正しい答えを，解答群の中から選べ。

a に関する解答群

ア ADDL GR4,GR5　　イ ADDL GR5,GR4　　ウ LD GR4,GR5　　エ LD GR5,GR4

オ SRL GR4,0,GR5　　カ SRL GR5,0,GR4

b に関する解答群

ア JMI ADDMXY　　イ JMI XMINUS　　ウ JMI YMINUS　　エ JPL ADDMXY

オ JPL XMINUS　　カ JPL YMINUS

c に関する解答群

ア JOV ADDMXY　　イ JOV SCHECK　　ウ JPL ADDMXY　　エ JPL SCHECK

オ JUMP ADDMXY　　カ JUMP SCHECK

d に関する解答群

ア SLL GR5,0,GR4　　イ SLL GR5,1　　ウ SRA GR5,0,GR4　　エ SRA GR5,1

オ SRL GR5,0,GR4　　カ SRL GR5,1

e に関する解答群

ア ADDL GR4,=1　　イ ADDL GR5,=1　　ウ SLL GR4,1　　エ SLL GR5,1

オ SUBL GR4,=1　　カ SUBL GR5,=1

設問2 プログラムを減算用に変更する場合，行番号 27 及び 32 の直後に追加する命令として正しい答えを，解答群の中から選べ。 なお，プログラム中のコメントは適宜読み替えるものとする。

解答群

ア AND GR2,=#7FFF　　イ AND GR2,=#8000　　ウ OR GR2,=#7FFF　　エ OR GR2,=#8000

オ XOR GR2,=#7FFF　　カ XOR GR2,=#8000

≪解答≫ 設問1 a：ウ　b：エ　c：カ　d：イ　e：ア　　設問2 カ

（基本 21 秋午後 12） 次のアセンブラプログラムの説明及びプログラムを読んで，設問1，2に答えよ。

〔プログラムの説明〕

連続したn語を 16 × n ビットのビット列とみなし，ビット列Aとする。ビット列Aの（p＋1）ビット目からのqビットを，別のqビットのビット列Bで置き換える副プログラム REPLACE である。置換えの概要を図1に示す。

ここで，p≧0，　1≦q≦16，　p＋q≦16×nとする。

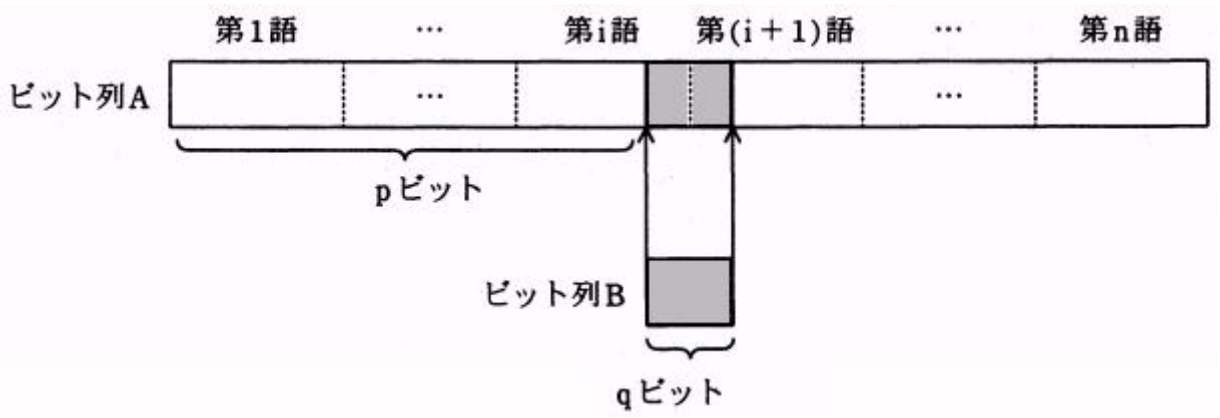


図1 置換えの概要

(1) ビット列Aの先頭アドレスは，GR1 に設定されて主プログラムから渡される。

(2) ビット列Bは GR0 に左詰めで設定され，GR0 の残りの部分は0で埋められて主プログラムから渡される。ビット列BとGR0 の関係を図2に示す。

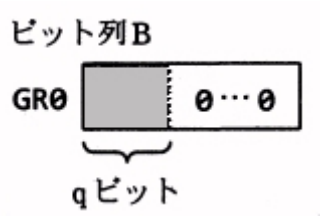


図2 ビット列Bと GR0 の関係

(3) 値pは GR2 に，値qは GR3 に設定されて主プログラムから渡される。

(4) 副プログラムから戻るとき，汎用レジスタ GR1～GR7 の内容は元に戻す。

〔プログラム〕

（行番号）

```
 1  REPLACE START
 2          RPUSH
 3          LD      GR4,GR2      ; GR4 ← p
 4          SRL     GR4,4        ; GR4 ← p/16
 5          ADDA    GR1,GR4      ; GR1 を置換え対象語(第 i 語)に位置付ける。
 6          AND     GR2,=#000F
 7          LD      GR4,=16
 8          SUBA    GR4,GR2
 9          LD      GR5,GR0
10          LD      GR6,=#8000
11          SUBA    GR3,=1
12          SRA     GR6,0,GR3
13          LD      GR7,GR6
14          SRL     GR0,0,GR2
15          SRL     GR6,0,GR2
16          SLL     GR5,0,GR4
17          SLL     GR7,0,GR4
18          LD      GR2,0,GR1
19          [ e ]   GR6,GR2      ; 第 i 語のうち
20          [ f ]   GR6,GR2      ; ビット列 B を入れる部分を 0 にする。
21          OR      GR2,GR0
22          ST      GR2,0,GR1
23          LD      GR2,1,GR1
24          [ e ]   GR7,GR2      ; 第(i+1)語のうち
25          [ f ]   GR2,GR7      ; ビット列 B を入れる部分を 0 にする。
26          OR      GR2,GR5
27          ST      GR2,1,GR1
28          RPOP
29          RET
30          END
```

設問1 次の記述中の［ a～d ］に入れる正しい答えを，解答群の中から選べ。

主プログラムから渡されたp，qの値及び GR0 の内容は，次のとおりであった。

p: 55

q: 12

GR0: 1011000111010000

行番号8の SUBA の実行直後における GR2 の値は, [a]であり, GR4 の値は [b]である。

行番号 17 の SLL の実行直後における GRO の内容は, [c]であり, GR5 の内容は[d]である。

a, b に関する解答群

ア 1　　イ 5　　ウ 7

エ 9　　オ 11　　カ 15

c,d に関する解答群

ア 0000000001011000　　イ 0000000101100011

ウ 1010000000000000　　エ 1110100000000000

設問2 プログラム中の[e～f]に入れる正しい答えを, 解答群の中から選べ。

解答群

ア ADDL　　イ AND　　ウ LD　　エ OR

オ SLL　　カ SRL　　キ XOR

≪解答≫ 設問1　(a) ウ　(b) エ　(c) イ　(d) ウ

設問2　(e) イ　(f) キ